DEWS2006 6B-oi3

# PC クラスタを用いた XML データ並列処理方式の提案

城戸健太郎† 天笠 俊之†,†† 北川 博之†,††

† 筑波大学大学院システム情報工学研究科 〒 305–8573 茨城県つくば市天王台 1–1–1
†† 筑波大学計算科学研究センター 〒 305–8573 茨城県つくば市天王台 1–1–1
E-mail: †kido@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

**あらまし** 近年，XML フォーマットの急速な普及と共に，当初は想定されていなかった，数百 MB から数 GB サイズの大規模なデータを XML で記述することが一般的に行われるようになってきている．このような巨大な XML データに対して，高信頼かつ高速なデータ格納，検索を行うには，XML データベースを利用するのが妥当な選択である．XML データベースを実現する主要な手法の一つに，関係データベースに XML データを写像することによって格納，検索を実現する，いわゆる関係 XML データベースがある．しかし，大規模な XML データでは処理性能が悪化することが指摘されている．この問題に対処するため，本論文では，PC クラスタによる XML データの並列処理方式を提案する．XML データの並列処理を可能にするため，まず XML データの分割方式について議論する．これは，XML スキーマグラフの部分分割および，対応する XML 部分インスタンスの集合分割に基づく，XML データの垂直，水平分割によって与える．これによって得られた XML データフラグメントのクラスタノードへの配置は，問合せ式（XPath）の処理コスト関数をもとに，各計算ノードのの処理コストを準最適化するような分割法をグリーディ法等によって求める．本論文ではさらに，提案手法の有効性を実験によって検証する．

**キーワード** XML, 並列・分散 DB，問合せ処理

# A Scheme for Parallel Processing of XML Data using PC Clusters

Kentarou KIDO†, Toshiyuki AMAGASA†,††, and Hiroyuki KITAGAWA†,††

† Graduate School of Systems and Information Engineering, University of Tsukuba
1-1-1 Tennoudai, Tsukuba, Ibaraki 305-8573, Japan
†† Center for Computational Sciences, University of Tsukuba
1-1-1 Tennoudai, Tsukuba, Ibaraki 305-8573, Japan
E-mail: †kido@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

**Abstract** Recently, with the rapid spread of XML format, it has become popular that large-scale data, whose size range from several hundreds of MB to several GB, are described by XML. For the purpose of providing fast and reliable means for storage and retrieval of huge XML data, it is a reasonable choice for us to use XML databases. In fact, there are many ways to realize XML databases, but relational XML database, in that an XML data is mapped to relational tables and query processing is enabled in terms of SQL queries, is one of the most popular way to implement XML databases. However, some researchers have pointed out that the performance of relational XML databases degrades when dealing with such huge XML data. In this study, we propose a scheme for parallel processing of XML data using PC Clusters. First, we discuss how to decompose XML data so that we can perform parallel processing of XML queries. We give the definitions of vertical and horizontal decomposition of XML data based on decomposition of schema graph and XML instances, respectively. To allocate decomposed XML data to cluster nodes, we give an algorithm for computing pseudo-optimal assignment of XML fragments by algorithms like greedy method in the light of XML query workload. Finally, we experimentally evaluate the effectiveness of the proposed method.

**Key words** XML, parallel and distributed DB, query processing

## 1. はじめに

XML（Extensible Markup Language）[1] は，データ記述フォーマットとして急速に普及し，関連技術の研究及び実装が盛んに行われている．XML データはタグの入れ子関係から任意の木構造を表現することが可能であり，タグ内の要素，属性に名前を持たせることで自己記述的な表現が可能である．最近では，大規模なデータを XML で記述することも多くなっており，数百 MB から数 GB のデータサイズを持つ大規模なものも少なくない．天文学での観測記録，バイオインフォマティクスにおける遺伝子データなどがその記述フォーマットの例として挙げられる．そのような背景から，大規模な XML データを効率的に扱うことのできる XML データベースが重要となってきている．

XML データベースの実現方法には何通りかの方法が提案されている．中でも関係データベースを利用する方法は，すでに稼動しているシステムが多数あることや，既存の情報資源との連携が取りやすいことなどから，主要な実現方法のひとつである．また，関係 XML データベースを構築する方法として，XML データをノード単位に分解し，それぞれをルートノードからの経路式とともに関係表にマッピングする手法（経路アプローチ）がある [2]．この手法の利点は，既存の問合せ最適化等の技術や，再帰的な問合せを効率的に処理することができる点である．商用システムでも Microsoft SQLServer 2005 がこの手法を採用している．しかし，関係データベースにおける XML データの処理はいくつかの理由から高コストであるため，大規模な XML データでは処理効率が悪化することが指摘されている [3].

そこで本研究では，大規模 XML データの高速な処理を目的として，PC クラスタによる XML データの並列処理方式を提案する．ここでの主な技術課題は，XML データを並列に処理する際，データをどのように分割し，どの PC クラスタノードに割り当てるかという XML データの分割と，計算ノードへの配置問題である．そこで本論文では，XML データの並列処理を可能にするため，まず XML データの分割方式について議論する．これは，XML スキーマの部分グラフの部分分割および，対応する XML 部分インスタンスの集合分割に基づく，XML データの垂直，水平分割によって与える．次に，ほとんどの XML 検索処理に XML データ全体は必ずしも必要ではないことに着目し，局所的に必要な部分をまとめておくことで処理の効率化を図る．具体的には，上の XML データ分割法を用いて，XML データを問合せワークロードの各問合せに必要なデータごとに分割する．これによって得られた XML データフラグメントは，問合せ式（XPath）のコスト関数をもとに，問合せワークロード全体の処理コストを準最適化するような配置法をグリーディ法によって求め，クラスタノードへ配置する．また，評価実験を行い，提案手法の有効性を確認する．

本論文の構成は次の通りである．第 2 章では関連研究について述べる．第 3 章では前提となる XML 関連技術について説明する．第 4 章では XML データの分割および分散配置の手法について説明し，第 5 章では評価実験について述べる．最後に第 6 章において，まとめと今後の課題について述べる．

## 2. 関連研究

まず，関係表の分割に関する研究として Anastassia 等の研究について述べる [4]．この研究は，関係データベースの自動チューニングに関する研究である．問合せワークロードが既知であると仮定し，関係表をグリーディ法により分割するアルゴリズムを提案している．これは問合せワークロードを用い，グリーディ法により関係表を分割している点が本研究に類似しているが，並列処理に着目した処理を考慮してない点，XML データの処理を考慮していない点が異なる．

次に，XML データの分割に関する研究として Brember 等の研究 [5] について述べる．これは，XML データの広域分散環境での格納，検索を可能にするための研究である．このため，Repository Guide と呼ばれる索引付けされた構造概要を利用している．まず，XML データの水平・垂直分割を XPath 式によって定義し，分割されたデータを Repository Guide によって管理する．Repository Guide は三種類の索引に関連付けられ，分散環境での問合せ処理を効率化している．この研究が本研究と本質的に異なる点は，分割された XML データがすでに分散環境に配置されていると仮定している点である．これに対して，本研究では XML データをワークロードを用いて動的に分割し，問合せ処理を最適化する手法を用いている．また，並列処理を考慮していない点が本研究と異なる．中尾等は XML データ内のアクセスの偏りを考慮し，問合せ処理が効率化されるように XML データを分割する手法を提案している [6]．XML データへのワークロードが既知であると仮定し，それらを元にアクセスの偏りと処理コストを考慮し，XML データをいくつかのフラグ目とに分割する．この研究はワークロードが既知であると仮定し XML データを分割している点，グリーディ法を用い準最適な分割法を提案している点が本研究に近いが，DOM や SAX による処理を対象にしている点や，厳密にコストモデルを定義していない点，また，並列処理を考慮してない点が本研究と異なる．夏目等は XML ファイルに意味を持たせて分割するアルゴリズムを提案すると共に，分割したファイルを自律ディスク上に格納し，履歴を用いて検索する手法を提案している [7].

## 3. 準　　備

### 3.1 XML データ

本研究で想定する XML データのモデルを示す．XML データ $D$ は $D=(V,E,r)$ で定義される．ここで，$V$ はノード集合，$E$ は枝集合，$r$ は根ノードであり $r \in V$ である．図 1 に XML データの例を示す．根ノードは XML データの最上位のノードである．a，b，c といった要素ノードは XML データの各要素

を表している．テキストノードは，要素の内容を表現するノードである．

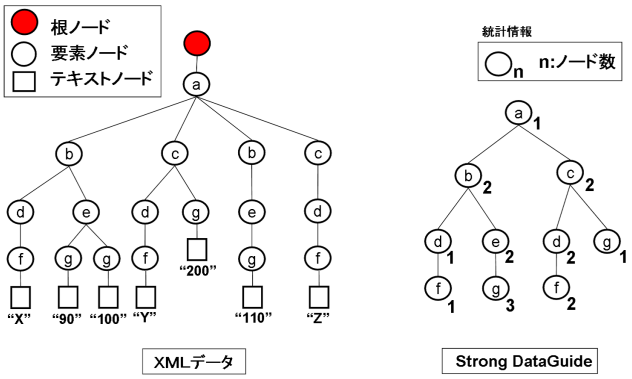


図 1　XML データと対応する DataGuide

### 3.2　DataGuide

XML データの問合せ処理を効率的に行うためには，XML データの全体の構造を知る手がかりが必要になる．このため，本研究では構造概要と呼ばれる半構造データのための索引構造を利用する．構造概要は，これまでにいくつかの方法が提案されている．本研究ではその中でも XML データを含む半構造データの構造をシンプルに木構造を用いて表現することができる Strong DataGuide [8] を用いる．Strong DataGuide とは，情報源において共通のラベル経路を持つノードを一つのノードに集約し，木構造表現したものである．なお，DataGuide では，情報源をラベル付きの木として用いているが，これを XML に適応する場合は，XML データの要素ノードの名前をラベルとみなす．定義等の詳細については文献 [8] を参照されたい．図 1 の左側に XML データを，右側に XML データから作成した Strong DataGuide ををを示す．例えば (a, b) といった XML データ中で共通のラベル系列を持つノードは，Strong DataGuide 中の (a, b) の一つのノードに集約されていることが分かる．以降特に断らない限り，DataGuide は Strong DataGuide を指すものとする．

本研究では構造概要に統計情報（ノード数）を格納し，4.2 節で述べる XPath 問合せのコストの計算の際に利用する．

### 3.3　XPath (XML Path Language)

本研究では問合せ言語として XPath [9] を用いる．XPath は 1999 年に W3C 勧告として公開された，XML データの特定の部分を指定するための汎用言語である．XPath では，さまざまな問合せ表現が可能であるが，その表現能力について，理論的な側面から評価した研究がある [10]．XPath の言語クラスは XP(L) を用いて表すことができ，L には記述可能な構文を表す．本研究では XP(/, //, []) を対象とする．これはノードテスト，親子関係を指定する child 軸，先祖子孫関係を指定する descendant 軸，および述語を含む問合せを記述することができることを意味する．

### 3.4　関係データベースへの XML データの格納

XML を関係データベースに格納する手法はこれまで多数提案されているが，XML データを経路式に基づき関係データベースにマッピングすることで，DTD や要素の型に依存することなく格納，検索することが可能となる．これを経路アプローチといい，代表的な手法として XRel [2] が挙げられる．この手法では XML データを解析して得られる木構造をノード単位に分割し，関係表に格納する．関係表に格納されたデータはもとの XML データに対してラベル付けを行うことにより，XML データとして再構築することが可能である．さらに，問合せ処理の高速化のために，B+木，R 木などの索引構造を利用することができる．本研究では XRel を参考に，XML データを NodeTable(did，pid, nodeid, nodenum, type, value)，PathTable(pid, pexp) の 2 つの表に格納する．関係表にマップされたデータは Dewey Order [11] に基づいてラベル付けをする．XML データを関係データベースに格納した例を図 2 に示す．

問合せ処理については，XPath を解析することによって得られる問合せグラフを SQL に変換することで，XPath 式を関係データベースの機能だけで評価することができる．問合せグラフとは，問合せを解析することによって得られるグラフ構造である．ここでは話を簡略化するため複雑でない問合せ（再帰的でない問合せ）について考える（図 3）．

まず，問合せを解析して得られる問合せグラフについて考える．問合せグラフはラベルノード，述語ノード，出力ノードの三つのノードからなる．ラベルノードは経路式によって表される．特に問合せ式中で [] によって表されたノードを述語ノードと呼ぶ．最終的に結果として出力される出力ノードであり，問合せグラフ中には必ず一つ，出力ノードが存在する．

問合せグラフを解析することにより，経路式を得ることができる．例えば，

```
//b[d/f='X']/e
```

といった問合せからは，

```
//b
//b/d/f
//b/d/e
```

の 3 つの経路式が得られる．

こうして得られた経路式から，次に示す項目を満たす用に SQL を生成する

- SELECT 項には出力ノードを表すテーブル名を記述.
- FROM 項には各ノードに対応するテーブル名を記述.
- SELECT 項では，まずそれぞれの問合せグラフの各ノードの経路式に対応するノードを，テーブルから絞り込むための条件を記述する.
- 述語ノードに関しては述語の内容を満たすように属性 val の条件を記述する.
- 最後にそれぞれのノードの親子関係を満たすように属性 nodenum を用いて条件を記述する.

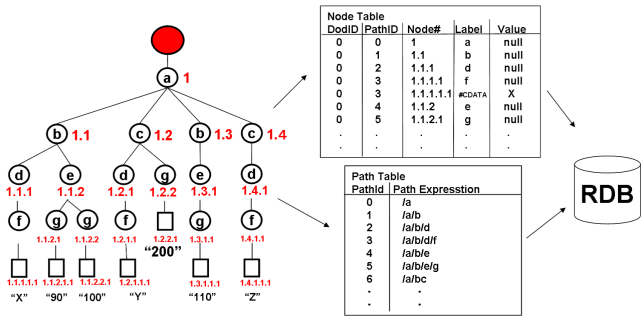


図 2　経路式に基づく関係表へのマッピング

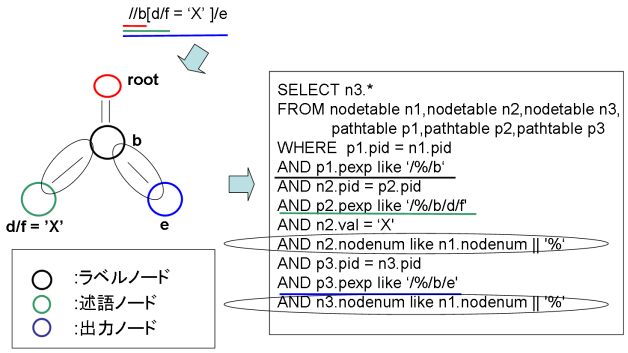


図 3　XPath から SQL の生成

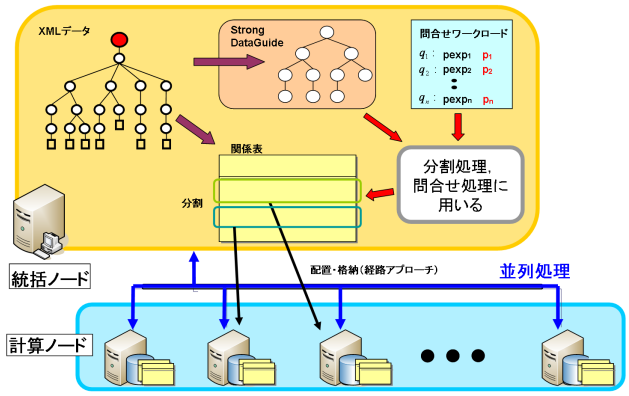


図 4　PC クラスタによる XML データの並列処理

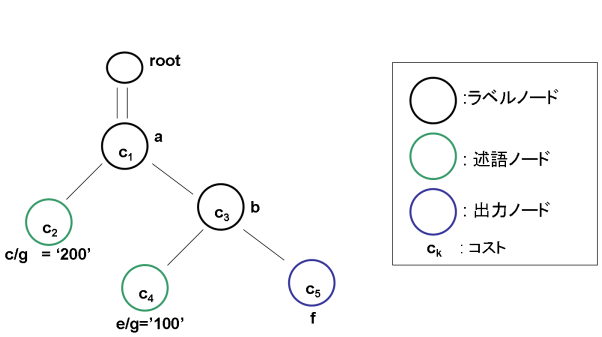


図 5　問合せグラフ

## 4.　提 案 手 法

### 4.1　システム構成

図 4 に本研究で提案するシステムの概要を示す．本研究では問合せ処理を管理する計算ノードとして統括ノードが存在することをを仮定する．XML データは初期状態では統括ノードのファイルシステムに格納されている．統括ノードは XML データを経路式にもとづき関係表をにマッピングする．その際，XML データの構造概要である DataGuide を生成する．DataGuide と問合せワークロードは，XML データの分割処理と問合せ処理に用いられる．関係表にマッピングされた XML データはワークロードに対する問合せコストが最小化されるように，分割される．生成されたフラグメントを計算ノードに配置し問合せ処理を行う．

ここで，本研究では，XML データに対するワークロード（問い合せセットと発行頻度）は既知であることを仮定する．発行頻度は問合せの発行履歴の統計から算出した確率を元に，上位 $n$ 件の問合せを典型的な問合せセットとし，その発生確率を合計が 1 になるように正規化した上で用いる．具体的には以下のように定義する

**定義** 問合せワークロード

問合せワークロード $W$ は，XPath 式 $q_i$ とその発行頻度 $r_i$ の対の集合 $W=\{(q_1,r_1),(q_2,r_2),\ldots,(q_n,r_n)\}$ とする．ただし $\sum_i^n r_i = 1.0$ である．なお，$W$ に含まれる全ての問合せを $W_q=\{q_1,q_2,\ldots,q_n\}$ で表すものとする．□

### 4.2　XPath 式のコスト算出

まず，提案手法を説明するのに必要な，XPath 式のコスト算出法を説明する．XPath 式 $q_k$ の処理コスト $C(q_k)$ は以下の式に基づいて計算する．

$$C(q_k) = a \cdot C_{retrieve} + (1-a) \cdot C_{reconstruction}$$

すなわち，関係表に対する問合せコスト $C_{retrieve}$ と，検索結果から XML データを再構築するコスト $C_{reconstruction}$ の和で表したものである．ここで，$a(a \leqq 1)$ は，$C_{retrieve}$ と $C_{reconstruction}$ の比を与える係数である．

$C_{retrieve}$ は 3.4 節で述べた問合せグラフを用いて次のように算出する．

（1） 問合せグラフを巡回し，各ノードが表す経路式に対応するタプルを関係表から選択するコスト $C_{select}$ を計算し，ノードに割り当てる．$C_{select}$ は選択アルゴリズムや索引の有無等の条件によって変わってくるが，ここでは関係表を走査するコストとして，関係表のサイズ（タプル数）$N$ を用いる．

（2） 述語ノードとその親ノードを構造結合（Structural Join）するコストを計算し，親ノードの $C_{select}$ を更新する．構造結合とは，各タプルに付与されているノード番号（本手法では Dewey Order）を利用し，候補タプルのうち XML データ上で実際に結合しているペアのみを残す操作である．構造結合のアル

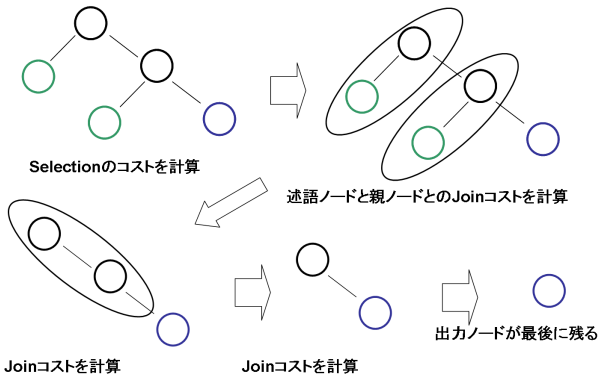


図 6　$C_{retrieve}$ の計算手順

ゴリズムとして Stack-Tree-Desc [12] があり，本研究ではこのアルゴリズムを用いる． Stack-Tree-Desc の詳細については，本論文の趣旨から外れるのでここでは説明しないが，そのコストは，先祖ノード数，子孫ノード数，結果ノード数の和で表すことができる．本手法では，結果ノードのサイズは知ることができないので，全ての候補が結果として残る最悪のケースを想定し，$C_{stj} = |A| + |B| + max(|A|, |B|)$ （$|A|$ は先祖のノード数，$|B|$ は子孫のノード数）として見積る．コストを算出し終わった後，述語ノードを削除して問合せグラフを縮退する．

（3） 最後に残ったノードの結合のコストを計算する．よく知られているように，$n$ 段の結合演算を効率的に実行するには，演算の適用順序の選択が重要である．どの隣接するペアを選ぶかは，上で述べた構造結合のコスト式 $C_{stj}$ より，先祖側のノード数に依存することが分かるので，各隣接するノードの組合せの中から，最も先祖側のノード数が少ないものを選ぶ．これまでと同様，次々とノードを縮退して行き，出力ノードが残るまで処理を繰り返す．残ったノードの $C_{select}$ が求める $C_{retrieve}$ である．

$C_{retrieve}$ は，出力ノードを根とする部分木を，関係表から XML データに再構築するコストのことである．これは，出力ノードの子孫ノードのデータ量から見積もることができるが，本研究では構造概要の統計値を用いて，該当するノード数を推定することで算出する．

### 4.3　提案手法の概要

提案手法の概要は以下の通りである．

（1） 問合せワークロードから 4.4 節で導入する分割法を用いて XML データをフラグメント化し，初期フラグメントを構成する．

（2） 次に，初期フラグメントを計算ノードに割り当てる．この際，上で導入したコスト関数によって推測される各計算ノードの負荷がなるべく均衡化するような割り当てを導出する．この問題は基本的には組合せ最適化問題なので，グリーディアルゴリズムや遺伝的アルゴリズム等を用いて準最適な割り当てを探索する．

以下では，まず XML データの分割法について説明する．

### 4.4　XML データの分割

関係データベースの分野では，主に分散環境での問合せ処理の効率化を目的として，関係表を分割した上で遠隔サイトに配置し，問合せを分散処理する手法が研究されてきた [13]．関係表を行方向に分割する手法を水平分割，列方向に分割する手法を垂直分割と呼ぶ．本研究では，同様の考え方に基き XML データの分散処理を試みる．

XML データの分割に関しては 2 節の関連研究で述べたように，Bremer 等の先行研究がある [5]．しかし，この研究で述べられている分割の定義では，完全性，排他性といった性質を保証することが難しい．完全性とは XML データをいくつかのフラグメントに分割したとき，それらを併合すると元の XML データを構成できるという性質である．また，排他性とは XML データを分割したとき，あるフラグメントは他のフラグメントと重複する要素を持たないという性質である．

そこで，本論文では XML データのスキーマグラフに着目し，スキーマグラフの部分分割と，得られた部分グラフに対応する（部分）XML インスタンスの集合分割を，それぞれ XML データの垂直，水平分割とすることを提案する．スキーマグラフとしては，XML インスタンスから導出した Strong DataGuide を用いる．

まず，スキーマグラフの形式的な定義から与える．XML データから導出される Strong DataGuide をスキーマグラフ $S = (V_s, E_s, r_s)$ と呼ぶ．ここで，$V_s$, $E_s$, $r_s$ はそれぞれノード集合，エッジ集合，$r_s \in V_s$ なるルートノードとする．$S$ を部分グラフに分割することによって，XML データの垂直分割を与える．

［定義 1］（垂直分割） $S = (V_s, E_s, r_s)$ をスキーマグラフとするとき，$F = \{(V_{f1}, E_{f1}), \ldots, (V_{fn}, E_{fn})\}$ なる $S$ の分割 $F$ を垂直分割と定義する．ただし，

- $V_s = \bigcup_{i=\{1,\ldots,n\}} V_{fi}$
- $i \neq j$ なる $fi, fj$ について $V_{fi} \cap V_{fj} = \emptyset$
- 任意の $fi$ について，$V_{fi}$ の相異なる 2 ノード $v, v' \in V_{fi}$ が $(v, v') \in E_s$ なら $(v, v') \in E_f$

を満たすものとする．また，$fi$ をフラグメント式，$fi$ に対応する（部分）XML インスタンスをフラグメントと呼ぶ．□

ここでは，対象とする XML データのスキーマグラフについて，全てのノードが包含される任意の分割を垂直分割であると定義している．このとき，元のスキーマグラフ上でエッジでつながれているノードは，分割された部分グラフ上でも接続されていることが条件となっている．部分グラフを重複がなく，かつ全ての要素が含まれるように設定することで，得られるフラグメントの完全性，排他性を保証できる．

図 7 に 垂直分割の例を示す．なお，フラグメント式の表記については，理解しやすいよう [5] で提案されている方法に準じるものとする．すなわち，部分グラフのルート要素の絶対経路式 $sf$ と，フラグメントから除外される部分グラフのルート要素への相対経路式 $ef_1, \ldots, ef_n$ を用いて，

$$f_{sf-\{ef_1,\ldots,ef_n\}}$$

のように表す．$sf$ は必須であるが，それ以外は必須ではない．

フラグメント式 $f$ が与えられると，それに対応する対応する XML データ集合が得られる．本研究では，ある垂直分割フラグメント式に対応する XML フラグメントの分割を水平分割と定義する．

［定義 2］（水平分割） $f=(V_f,E_f)$ を $S$ の垂直分割フラグメント式とし，$D=\{d_1,d_2,\ldots,d_n\}$ を $f$ のフラグメントとする．このとき，$D=\bigcup_i D_i, D_i \subseteq D$ なる $D$ の分割を $f$ の水平分割と呼ぶ．□

なお，水平分割 $D_i$ の表記は，これに含まれる全てのフラグメントが満たす条件を XPath 式を使って現すことにする．このとき，与える条件を適切に設定することで，フラグメントが完全性，排他性を満たすように分割を行うことができる．

図 8 は，垂直分割のうち左下のフラグメントの水平分割の例 $f_{/a/b[e/g<100]}$, $f_{/a/b[e/g\geq 100]}$ である．分割に与える条件を適切に設定することで，重複するフラグメントを持たない分割を定義することができる．

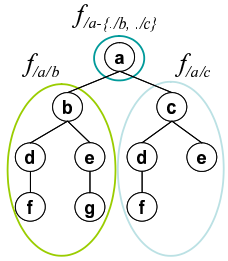


図 7　XML データの垂直分割

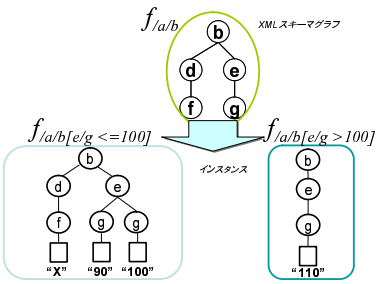


図 8　XML データの水平分割

### 4.5　初期フラグメントの構成

提案手法では，まずワークロードに含まれる経路式を元に，初期フラグメントを構成する．まず，ワークロード $W=\{(q_1,r_1),(q_2,r_2),\ldots,(q_n,r_n)\}$ の XPath 式から，前述の方法で全ての経路式を抽出する．述語を含む問合せの場合は，二つ以上の経路式が抽出されるので，$q_i$ から抽出される経路式を $p_{i1},\ldots,p_{ik}$ とする．基本的には，各 $p_{ik}$ に対応する $f_{p_{ik}}$ と，ワークロードに含まれないノードに対応するフラグメント式（$f_c$ とする）を元に垂直分割を構成する．なお，ワークロードによっては，いくつかの経路式が重複する部分を持つ場合があるが，その場合は，図 7 に示したように，重複が起こらないような分割を求める．

各フラグメントのサイズ，すなわち対応する XML ノードの数は，DataGuide に格納されている統計値（ノード数）から知ることができる．この値は，コスト計算を行う際に用いられる．フラグメント $f_p$ のサイズは $size(f_p)$ で記述するものとする．

### 4.6　初期フラグメントの計算ノードへの配置

一般に，初期フラグメントの数は計算ノードの数より多い．また，フラグメントによってはワークロード中の使用頻度が大きく異なることがあるので，各計算ノードの負荷がなるべく均等になるように初期フラグメントを計算ノードに配置する．各計算ノードの負荷は，ワークロード中の XPath 式の処理コストをフラグメント単位に分けて考え，対応する計算ノード毎に合計した値を用いる．積算の際には，発生頻度によって重み付けを行う．

各計算ノードの負荷を最も下げるような最適なフラグメントの割り当てを全数探索で求めるには，計算ノード数を $N$，フラグメント数を $m$ とすると $O(m^N)$ のコストがかかり現実的ではない．そこで，以下に示すようなアルゴリズムを適用して準最適解を求める．

**グリーディアルゴリズム**　初期フラグメントを適当な方法（ラウンドロビンやランダム等）で計算ノードに割り当て，これを初期割り当てとする．次に，各ノード毎に負荷を計算し，最も負荷の高いノードについて，割り当てられているフラグメントをその他の（負荷の低い）ノードに仮に割り当て，コストを再計算する．その中でも最も良い割り当て方法を選択し，値が改善されなくなるまでこれを繰り返す．

**遺伝的アルゴリズム**　アルファベットを各計算ノードのシンボルとする長さ $m$ の遺伝子を考え，これを元に遺伝的アルゴリズムによって各計算ノードの負荷が均衡化するような最適解を探索する．

## 5.　予 備 実 験

上で述べた提案手法によって，処理性能が改善するかどうかを検証するために，予備実験を行った．

### 5.1　実 験 環 境

用いた計算機のスペックは，統括ノードについては表 1，計算ノードについては表 2 の通りである．また，計算ノードは 5 台用いた．

表 1　実験環境（統括ノード）

| | |
|---|---|
| CPU | Intel(R) Xeon(TM) 3.0GHz x 4 |
| OS | Red Hat Enterprise Linux 4.0 |
| メモリ | 6GB |
| JAVA | J2SE 1.5 |

表 2　実験環境（計算ノード）

| | |
|---|---|
| CPU | Intel(R) Xeon(TM) 3.0GHz x 4 |
| OS | Red Hat Enterprise Linux 3.0 |
| メモリ | 1GB |
| DB | PostgreSQL 8.1.0 |

表 3　問合せセットとそれらの発行頻度

| 問合せ | 問合せ式 | 発行頻度 |
|---|---|---|
| $q_1$ | //mail[date = ’10/20/2000’]/text | 0.3 |
| $q_2$ | //person[@id = ’person23’]/address | 0.25 |
| $q_3$ | //date | 0.2 |
| $q_4$ | //regions/asia | 0.15 |
| $q_5$ | //person/profile[/age = ’23’] | 0.1 |

データセットは XMark プロジェクト [14] で公開配布されている XML データ生成プログラム xmlgen によって 100K，1M，10M，100M のデータを生成し，4 章で述べた手法により分割，計算ノードへの配置を行った．

### 5.2 実 験 方 法

実験では，表 3 のワークロードに従う 100 個の問合せを生成し，次の 4 つの場合について処理時間を比較した．

（1） XML データの分割を行わず，1 ノードで処理をした場合

（2） XML データを分割し，1 ノードで処理をした場合

（3） XML データを分割し，ワークロードを考慮せず配置した場合

（4） XML データを分割し，提案手法で配置した場合

### 5.3 実 験 結 果

表 4　実験結果 [msec]

| | (1) | (2) | (3) | (4) |
|---|---|---|---|---|
| 0.1MB | 925 | 921 | 4867 | 212962 |
| 1MB | 1143 | 8155 | 4215 | 98292 |
| 10MB | 1612 | 1954 | 8146 | 144927 |
| 100MB | 1057 | 6380 | 3164 | 47663 |

実験結果を表 4，表 4 をグラフにしたものを図 9 に示す．グラフについては，縦軸は処理時間 [ms] を，横軸はデータサイズ [MB] をそれぞれ表す．

まず，全く分割を行わなかった場合より，分割を行い処理をした場合（1 ノード）について問合せ処理の高速化が見られた．これは，分割することによって関係表のサイズが小さくなり，データ読み込み時間が低減されたためと考えられる．

分割したデータを，ワークロードを考慮せず各計算ノードに割り当てた場合は，1 ノードで処理をした場合より処理が遅くなった．本研究では，各ノードに処理に必要なデータがそろっている場合は，JOIN の処理を各ノードに任せ，もしそろっていない場合は統括ノードで JOIN の処理を行う．よって，多くの JOIN の処理を統括ノードで行わなければならないため，実行効率が低下したものと考えられる．

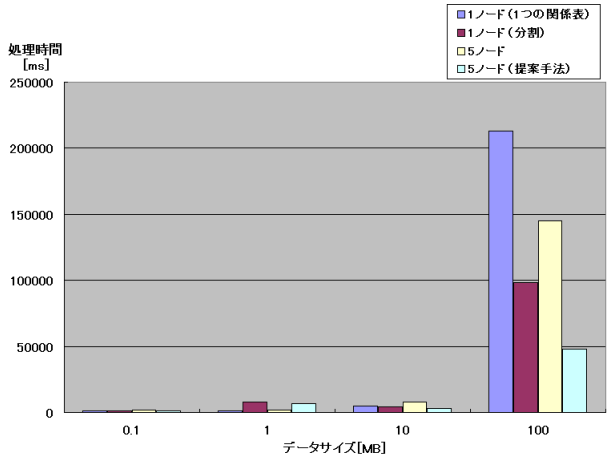


図 9　実 験 結 果

提案手法による結果は 1 ノードで分割したデータを処理をした場合に比べて，約 50%処理効率が改善されている．これは提案手法により，ワークロードを考慮し分割したデータを計算ノードに割り当てたため，全体の処理の並列性が上がったためであると考えられる．

データサイズが小さい場合は性能にあまり違いがでないが，データサイズが大きい場合（本稿では 100MB）では顕著な違いが見られた．

## 6. ま と め

本研究では PC クラスタを用いた XML データ並列処理方式を提案した．XML データの並列処理を可能にするため，まず XML データの分割方式を定義した．これは，XML スキーマグラフの部分分割および，対応する XML 部分インスタンスの集合分割に基づく，XML データの垂直，水平分割によって与える．これによって得られる XML データフラグメントのクラスタノードへの配置は，問合せ式（XPath）の処理コスト関数をもとに，問合せワークロード全体の処理コストを準最適化するような分割法を求める．さらに，提案手法の有効性を実験によって検証した．

今後はさらにシステムの実装と性能評価を行い，また，問合せの並列性を高めるための，XML データ分割方式を検討する予定である．また，XML データのアップデート機能の検討などが今後の課題として挙げられる．

## 謝　　辞

本研究の一部は，科学研究実費補助金，基盤研究 (B)(#15300027)，若手研究 (B)(#17700110) および科学技術振興機構戦略的創造研究推進事業 CREST の支援により行われた．

## 文　　献

[1] World Wide Web consortium: Extensible Markup Language (XML) 1.0 (Third Edition), http://www.w3.org/TR/REC-xml. W3C Recommendation 04 February 2004.

[2] M. Yoshikawa, T. Amagasa, T. Shimura and S. Uemura: “XRel: A

path-based approach to storage and retrieval of XML documents using relational databases", ACM Transactions on Internet Technology(TOIT), **1**, 1, pp. 110–141 (2001).

[3] 天笠俊之, 植村俊亮："リージョンディレクトリを用いた関係データベースによる大規模 XML データ処理", 日本データベース学会 Letters, **3**, 2, pp. 33–36 (2004).

[4] S. Papadomanolakis and A. Aliamaki: "AutoPart: Automatin Schema Design for Large Scientific Databases Using Data Partitioning", Proc. SSDBM (2004).

[5] J. Bremer and M. Gerts: "On Distributing XML Repositories", In 6th International Workshop on the Web and Databases WebDB2003, pp. 73–78 (2003).

[6] 中尾伸章, 天笠俊之, 的野晃整, 植村俊亮："アクセス頻度を考慮した XML 文書分割方式の提案 ", Proc. DEWS2005 (2005).

[7] 夏目亘, 横田治夫："分散ディスクへの XML データの分割格納方法", Proc. DEWS2001 (2001).

[8] R. Goldman and J. Widom: "DataGuides: Query Formulation and Optimization in Semistructured Databases", Proc. ICDE (2002).

[9] World Wide Web consortium: XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/1999/REC-xpath-19991116. W3C Recommendation 16 November 1999.

[10] T. Schwentick: "XPath Query Containment", ACM SIGMOD Record (2004).

[11] Online Computer Libraly Center. Introduction to the Dewey Decimal Classification, http://www.oclc.org/dewey/versions/abridgededition14/intro.pdf.

[12] S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava and Y. Wu: "Structural Joins: A Primitive for Efficient XML Query Pattern Matching", Proc. VLDB (1997).

[13] M. T. Ozsu and P. Valduriez: "Principles of Distributed Database Systems(2nd Edition)", Prentice Hall (1999).

[14] An XML Benchmark Project (XMark). http://monetdb.cwi.nl/xml/index.html, 2003.